# Comparing MNIST digits with weight sharing and auxiliary losses

Ridha Chahed, Ghassen Karray, Haitham Hammami
EPFL EE-559  DEEP LEARNING

***Abstract*— MNIST digit classification is a classic Convolutional Neural Network problem. In this project we will be using the MNIST dataset to try out different architectures of neural nets using PyTorch in order to predict a comparison of two handwritten digits.**

## I. Introduction

The objective of this project is to recognize and compare two handwritten digits visible in a two-channel image. More specifically we are trying to test different models and benchmark their corresponding performances, in particular we will investigate the impact of integrating Siamese Network into the architecture while adding weight sharing and auxiliary losses.

## II. Data Generation

The data consists of 2 datasets each of 1000 pairs of 14×14 grey-scale images. We have built a generator that returns the train and test datasets which consist of pairs of images, their corresponding classes and the result of the comparison, the latter has the value 0 if the first image is strictly bigger than the other, and 1 otherwise.

## III. Model construction

### A. Initial parameters

We started by defining the skeleton of our model, which consists of:

- Loss function: We chose to work with *Binary Cross Entropy* since we are dealing with a binary classification problem.
- Activation functions: For the hidden layers, we initially chose *Leaky ReLU* to remedy the vanishing gradient problem. As for the output, *Sigmoid* was a natural choice to get a prediction between 0 and 1.
- Optimizer: We opted for *Adam* as optimizer as it proved to be a robust optimizer. Initially we set the learning rate to be 0.001 with a decay rate of 0.9 per epoch.
- Dropout: Which is a technique used to prevent overfitting by applying a layer that randomly drops a proportion of units along with their connections during training. We chose 0.2 as a fraction of the dropout.
- Number of Epochs: We considered 20 epochs to be sufficient as our data isn't very diverse.
- Batch Size: Since the data isn't very large, setting the batch size to 5 gave an acceptable computational duration.

### B. Basic Net

In a first approach, we built a simplistic neural network that is made from a hidden layer of 512 nodes and an output layer. After training this model we get an average accuracy of the test set of 0.714 and a standard deviation of 0.014 that we will use as a baseline in our later research.

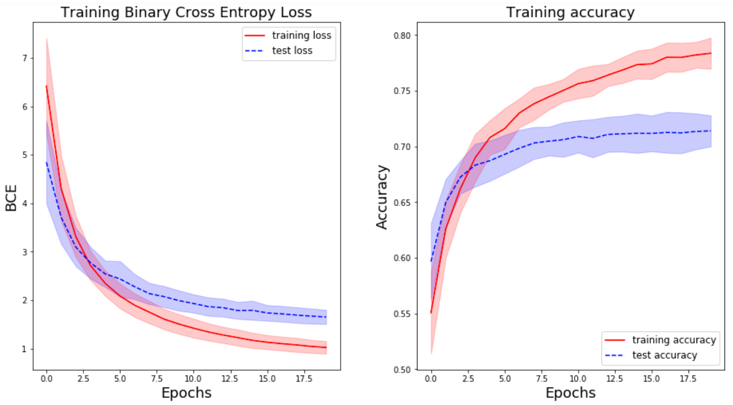

***Fig. 1:*** *Basic Net*

### C. *Siamese Network*

A Siamese network consists of two identical sub-networks that are joined at their output. In addition two that we will integrating two important features of Siamese subnets:

- Weight Sharing between the Siamese Networks, which consists of having the same tensor of weights for the two sub-networks and therefore adjusting its weights to the two losses at the same time.
- Auxiliary Losses are losses incurred at the end of the sub-nets and propagated to the final loss. The two auxiliary losses are added to the output loss as follows :

  $$Total = \alpha(Output) + \frac{1-\alpha}{2}(Left + Right) \tag{1}$$

  This is helpful in our case since it offers the sub-nets a better insight of the loss that was directly caused by them during back-propagation.

Table I is a modularisation of two different and customizable Siamese networks with the following architectures:

| **CNN** | **FCN** |
|---|---|
| Input: 14 x 14 | Input: 14 x 14 |
| Conv2d: **BC** x **K**<br>Leaky ReLU<br>MaxPool: 2 x 2<br>Dropout: 0.2<br><br>Conv2d: (2***BC**) x **K**<br>Leaky ReLU<br>Dropout: 0.2<br><br>Linear: **OC** x **HL**<br>Leaky ReLU<br>Dropout: 0.2<br><br>**KL** - 1 times :<br>Linear: **HL** x **HL**<br>Leaky ReLU<br>Dropout: 0.2 | Linear: 14*14 x **HL**<br>Leaky ReLU<br>Dropout: 0.2<br><br>**KL** - 1 times :<br>Linear: **HL** x **HL**<br>Leaky ReLU<br>Dropout: 0.2 |
| Linear: **HL** x 10 | Linear: **HL** x 10 |

***TABLE I:*** *FCN and CNN parameterization.*
***BC****: base channel size -* ***K****: kernel size -* ***KL****: Number of hidden layers -* ***HL****: Units in hidden layer -* ***OC****:Units in output convolutional layer*

## IV. Results and discussions

### A. *Initial results*

We started by setting $HL = 64$ and $KL = 1$ for the FCN, $BC = 4$, $K = 3$,$HL = 64$ and $KL = 1$ for the CNN and $\alpha = \frac{1}{2}$ for both auxiliary losses. We tested our model with and without auxiliary loss and weight sharing and table II summarises our findings when we run for 10 rounds and compute the score as the mean accuracy after 20 epochs of training. We observe a net improve of the accuracy on the test set when we add the auxiliary loss and the weight sharing for the two subnet architectures. We make the hypothesis that we can improve these scores by tuning the hyper-parameters especially for the convolutional subnet that has many.

| Weight Sharing | Auxiliary Loss | FCN Score | FCN Std | CNN Score | CNN Std |
|---|---|---|---|---|---|
|  |  | 0.828 | 0.006 | 0.833 | 0.158 |
|  | ✓ | 0.870 | 0.007 | 0.882 | 0.011 |
| ✓ |  | 0.851 | 0.008 | 0.864 | 0.006 |
| ✓ | ✓ | 0.892 | 0.006 | 0.902 | 0.013 |

***TABLE II:*** *Results of FCN and CNN*

### B. *Hyperparameter tuning*

After fixing the general outline of our model, now we seek to improve furthermore its accuracy by applying a grid search on its hyper-parameters in order to find the optimal values. Table III and IV provide a summary of our findings.

| Parameter | Values | Best | Score |
|---|---|---|---|
| KL | [1,2] | 2 | - |
| HL | [32,64,128,256,512] | 128 | 0.9105 |
| C-KL | [1,2] | 2 | - |
| C-HL | [32,64,128,256,512] | 256 | 0.9205 |
| $\alpha$ | [0:1] | 0.11 | 0.9282 |

***TABLE III:*** *Optimization results of FCN.*
***KL****: Number of hidden layers -* ***HL****: Units in hidden layer -* ***C****: refers to layers after the combination of the subnets -* $\alpha$*: auxiliary loss fraction*

| Parameter | Values | Best | Score |
|---|---|---|---|
| KL | [1,2] | 2 | - |
| HL | [32,64,128,256,512] | 512 | 0.9298 |
| BC | [4,8,16,24,48] | 24 | 0.9638 |
| K | [3,5] | 3 | - |
| C-KL | [1,2] | 2 | - |
| C-HL | [32,64,128,256,512] | 512 | 0.9676 |
| $\alpha$ | [0:1] | 0.44 | 0.9721 |

***TABLE IV:*** *Optimization results.*
***BC****: base channel size -* ***K****: kernel size -* ***KL****: Number of hidden layers -* ***HL****: Units in hidden layer -* ***C****: refers to layers after the combination of the subnets -* $\alpha$*: auxiliary loss fraction*

For the fully connected subnet we observe that the models without auxiliary loss tend to overfit on the training set and thus give worst results on the test set. On the other hand, the weight sharing doesn't significantly affect the accuracy on train set but improves it on the test set.

We obtain better results with the convolutional network subnet (figure 2) as we increased the capacity of the network so it has less tendency to overfit. We finally obtain an average accuracy of **0.97**. The summary of the final architecture is depicted in figure 3

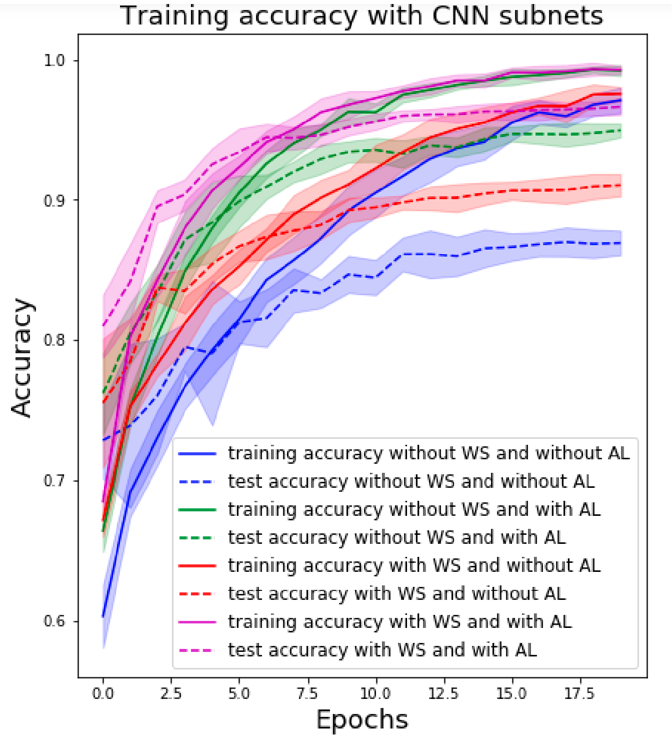

***Fig. 2:*** *Optimal CNN subnet*

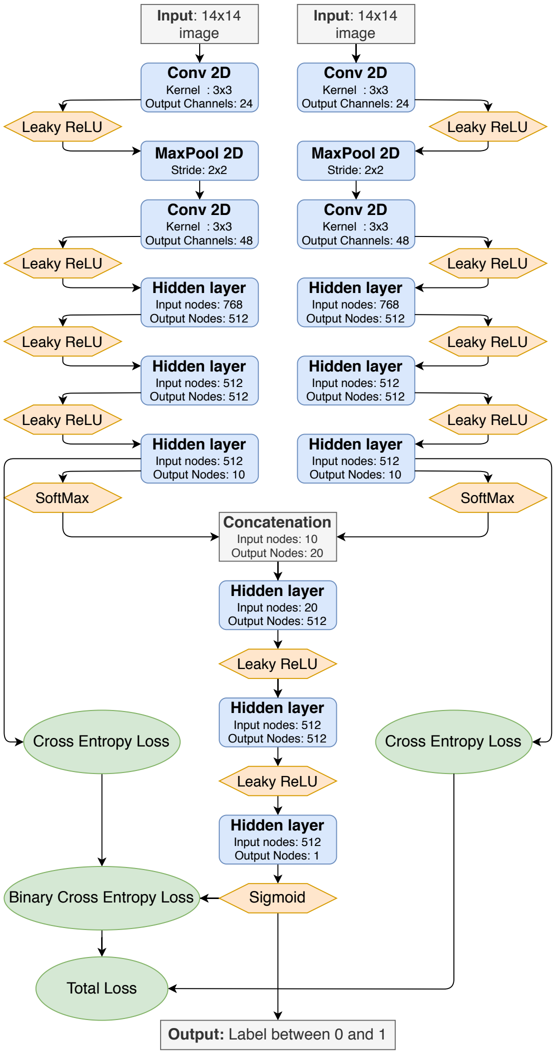
Input: 14x14 image
Input: 14x14 image
Conv 2D Kernel : 3x3 Output Channels: 24
Conv 2D Kernel : 3x3 Output Channels: 24
Leaky ReLU
Leaky ReLU
MaxPool 2D Stride: 2x2
MaxPool 2D Stride: 2x2
Conv 2D Kernel : 3x3 Output Channels: 48
Conv 2D Kernel : 3x3 Output Channels: 48
Leaky ReLU
Leaky ReLU
Hidden layer Input nodes: 768 Output Nodes: 512
Hidden layer Input nodes: 768 Output Nodes: 512
Leaky ReLU
Leaky ReLU
Hidden layer Input nodes: 512 Output Nodes: 512
Hidden layer Input nodes: 512 Output Nodes: 512
Leaky ReLU
Leaky ReLU
Hidden layer Input nodes: 512 Output Nodes: 10
Hidden layer Input nodes: 512 Output Nodes: 10
SoftMax
SoftMax
Concatenation Input nodes: 10 Output Nodes: 20
Hidden layer Input nodes: 20 Output Nodes: 512
Leaky ReLU
Cross Entropy Loss
Hidden layer Input nodes: 512 Output Nodes: 512
Cross Entropy Loss
Leaky ReLU
Hidden layer Input nodes: 512 Output Nodes: 1
Binary Cross Entropy Loss
Sigmoid
Total Loss
Output: Label between 0 and 1

***Fig. 3:*** *Model architecture*

## V. Conclusion

The MNIST dataset was and still is a robust example for testing and exploring the abilities of neural networks for image recognition. In particular here we were able to prove how much of a powerful tool neural nets can be for image comparison through the utilization of Siamese Networks, especially when combined with weight sharing and auxiliary losses. These two features allowed us to increase the network capacity to generalize the ability of recognition, giving a better accuracy score on the test set . We also concluded that a fully connected network can give a very good performance on its own. Nevertheless it cannot beat the performance of a well-optimized convolutional neural net when it comes to image recognition